3D Fabrication with Universal Building Blocks and Pyramidal Shells

XUELIN CHEN*, Shandong University
HONGHUA LI*, Alibaba AI Labs
CHI-WING FU, The Chinese University of Hong Kong
HAO ZHANG, Simon Fraser University
DANIEL COHEN-OR, Tel Aviv University
BAOQUAN CHEN, Peking University and Shandong University

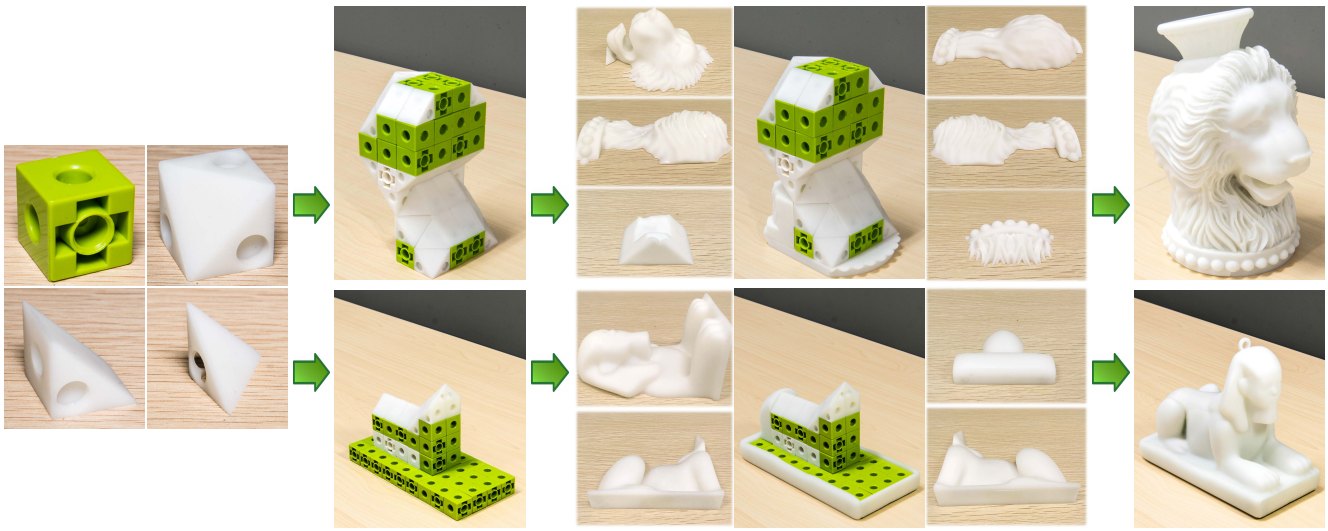


Fig. 1. We introduce a method for cost-efficient 3D fabrication by decomposing a 3D shape into (i) an internal core, to be assembled from universal building blocks, and (ii) the residual, to be further decomposed into approximately pyramidal shells for efficient 3D printing. From left to right, the four types of blocks, internal cores assembled from the blocks, 3D-printed pyramidal shells forming the residual, and the final object assembly. The two 3D objects clearly have very different shapes, yet their internal cores can be assembled from exactly the same set of universal building blocks.

We introduce a computational solution for cost-efficient 3D fabrication using *universal building blocks*. Our key idea is to employ a set of universal blocks, which can be massively prefabricated at a low cost, to quickly assemble and constitute a significant *internal core* of the target object, so that only the *residual volume* need to be 3D printed online. We further improve the fabrication efficiency by decomposing the residual volume into a small number of printing-friendly pyramidal pieces. Computationally, we face a *coupled decomposition* problem: decomposing the input object into an internal core and residual, and decomposing the residual, to fulfill a combination of objectives for efficient 3D fabrication. To this end, we formulate an optimization that jointly minimizes the residual volume, the number of pyramidal residual pieces, and the amount of support waste when printing the residual pieces. To solve the optimization in a tractable manner, we start with a maximal internal core and iteratively refine it with local cuts to minimize the cost function. Moreover, to efficiently explore the large search space, we resort to cost estimates aided by pre-computation and avoid the need to explicitly construct pyramidal decompositions for each solution candidate. Results show that our method can iteratively reduce the estimated printing time and cost, as well as the support waste, and helps to save hours of fabrication time and much material consumption.

CCS Concepts: • **Computing methodologies → Shape modeling**;

Additional Key Words and Phrases: fabrication, building blocks, decomposition, 3D printing, cost efficiency

ACM Reference Format:
Xuelin Chen, Honghua Li, Chi-Wing Fu, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. 2018. 3D Fabrication with Universal Building Blocks and Pyramidal Shells. *ACM Trans. Graph.* 37, 6, Article 189 (November 2018), 15 pages. https://doi.org/10.1145/3272127.3275033

*Both authors contributed equally to the paper

Authors' addresses: Xuelin Chen, Shandong University; Honghua Li, Alibaba AI Labs; Chi-Wing Fu, The Chinese University of Hong Kong; Hao Zhang, Simon Fraser University; Daniel Cohen-Or, Tel Aviv University; Baoquan Chen, Peking University and Shandong University.

0730-0301/2018/11-ART189 $15.00
https://doi.org/10.1145/3272127.3275033

1 INTRODUCTION

Recent advances in 3D printing technologies and their growing application potential have attracted much interest in the computer graphics community. Using this additive manufacturing technique, one can build a target object layer by layer, enabling the production of 3D objects of almost any shape. However, 3D printing remains a highly time-consuming process with costly material consumption [Bermano et al. 2017], especially when the entire volume of a large object, and possibly support waste, both need to be printed.

One major approach to cost-efficient 3D fabrication is to print only a *portion* of the target object; the rest is either left as hollowed in the object's interior [Lu et al. 2014; Stava et al. 2012; Wang et al. 2013; Zhang et al. 2015] or built with a different material. This latter approach can directly overcome the intrinsic limitations of 3D printing and open the door to innovative fabrication alternatives for cost reduction [Mueller et al. 2014; Song et al. 2016]. However, to maximize the benefit of such approaches, one must face a multi-objective optimization problem. On the one hand, the volume of the 3D printed portion and the associated support waste must both be minimized. On the other hand, the resulting (relative) increase in the remaining portion of the object must be well compensated by a clever fabrication alternative and material selection.

In this paper, we introduce a novel approach for cost-efficient 3D fabrication, which complements 3D printing with a fabrication alternative based on *universal building blocks* that can be *massively prefabricated*. Specifically, we decompose the target 3D object into an *internal core* that is assembled by the building blocks, while only the remaining portion (referred to as the *residual volume*) is 3D printed; see Figure 1. Considering the overall fabrication efficiency, for both processing time and material consumption, our approach hinges on two key criteria. First, the building blocks must be made up of *low-cost* material, easy to assemble, and universal. Second, 3D printing of the residual volume must be efficient. This is dictated by the printing strategy and the shape of the internal core.

To cost-effectively 3D print the residual volume, we decompose it into a small number of *printing-friendly residual parts*. To this end, we target *pyramidal* shapes [Hu et al. 2014] for these parts. A pyramidal shape is a height field over a flat base and can be 3D printed using an FDM printer without any support waste. Furthermore, to reduce the number of residual parts, it is desirable for the internal core to have a small number of flat surfaces, which can serve as the flat bases of the pyramidal (residual) parts. This consideration is correlated with our choice of the building blocks.

In general, besides universality, the building blocks should also be *tileable*, so that they can be assembled into a wide range of 3D shapes to serve as the internal core. In addition, the building blocks should be geometrically simple to allow economical construction and easy assembly, without compromising the printability of the residual. With these criteria in mind, we choose four types of building blocks: a cube and three sub-volumes of the cube whose faces are all parallel to the principal axes or main diagonals of the cube; see Figure 1. With cubes as the principal building blocks of the internal core, the other pieces can help to fill the concavities around the cubes, further reducing the residual volume while allowing flat surfaces of more orientations to serve as the bases of the residual parts.

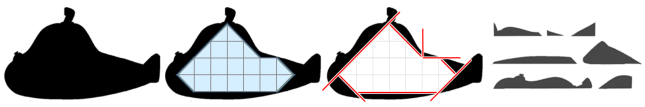


Fig. 2. A 2D illustration of our decomposition. From left to right, the input shape is decomposed into an internal core (in light blue), which is to be assembled from 2D versions of the building block set, and a residual shell volume (in black), which is to be further decomposed into approximately pyramidal pieces for online printing.

The core computational problem involves *two* decompositions: (i) how to decompose the input object into an internal core and the residual, and (ii) how to decompose the residual into pyramidal shapes for 3D printing; see Figure 2. The main challenge is that the two decompositions are *coupled*, meaning that we should not consider them independently. Using the best geometric approximation of the input object as the internal core is ill-advised, since the residual volume would be too thin with complex bases, thus compromising its printability. Hence, we need to carefully trade-off between minimizing the residual volume and maximizing the printability of the residual parts.

We develop a computational approach by formulating the coupled decomposition with an objective function accounting for the size of the residual, residual part count, and amount of support waste. Since the resulting optimization problem is highly non-linear and induces a tremendous search space, we resort to a heuristic for efficiency. Specifically, we initialize the solution with a maximal internal core, and employ beam search to iteratively refine the internal core, with *local cuts* aligned with the orientations of the building blocks to minimize the objective function. Moreover, since it is overly costly to evaluate the exact residual part count and amount of support waste through an explicit pyramidal decomposition of the residual for every single solution candidate, we develop efficient estimates for these terms in the objective function to boost the performance. Lastly, for the aesthetics of the results, we avoid cutting salient regions over the object when decomposing the residual volume.

To the best of our knowledge, our method represents the first computational framework for a hybrid, high-fidelity 3D fabrication using universal building blocks. It only requires building blocks of a few types (four per scale to be exact). These blocks are independent of the shape of the target 3D objects and they can be massively pre-manufactured using traditional techniques such as molding. Just as importantly, our building blocks are universal and geometrically simple, so we can quickly assemble them for a wide range of 3D shapes, which are not necessarily convex or bulky. Figure 1 shows an example of using the *same* block set to fabricate two drastically different 3D objects. Also, by selecting and mixing blocks at suitable scales, our method is effective for fabricating objects, large or small. Quality-wise, the surfaces of the final 3D objects are reproduced to printer precision. We demonstrate our method over a rich variety of shapes and estimate the overall fabrication time and cost using Cura [Ultimaker ltd. 2017] to compare its efficiency against conventional 3D printing. Results show that our method saves hours of 3D printing time and much material consumption, while taking only minutes to compute for most tested objects; see Section 6.

2 RELATED WORK

A key idea to improve onsite construction efficiency is the use of universal building blocks that are massively prefabricated [Wikipedia 2016]. The important characteristic is that the building blocks are *not customized* to one specific construction. In this section, we mainly focus on related works for cost-effective 3D printing.

Hollowing before printing. A typical strategy for cost-effective 3D printing is to hollow an object's interior and *custom*-build an internal structure to support the 3D-printed shell, e.g., by inserting inner struts [Stava et al. 2012] or by constructing optimized structures such as those involving skin-frames [Wang et al. 2013], honeycomb-cells [Lu et al. 2014], or medial axis trees [Zhang et al. 2015]. Other methods optimize the interior void for other purposes, e.g., to balance the printed object [Prévost et al. 2013] or to fulfill prescribed buoyancy constraints [Wang and Whiting 2016]. An interesting variation of the latter work is to use off-the-shelf metal bolts or cement to fill parts of the interior volume. There is a slight flavor of prefabrication, but the cheap material is only for filling a *given* volume. Instead of carving out an internal volume to reduce the filling during 3D printing, our method computes an internal core to maximize the benefit of using universal building blocks, leading to an entirely different optimization. Since the cost of 3D printing grows cubically as the volume of the target object scales up, the larger the object, the more cost saving is gained by substituting the object interiors with prefabricated building blocks.

Decomposition. Another strategy for cost-effective fabrication is to decompose an object into pieces that are more printing-friendly, or can be packed into the printer volume and printed simultaneously to save the overall fabrication cost [Chen et al. 2015; Vanek et al. 2014; Yao et al. 2015]. Early work by Luo et al. [2012] decomposes an object, so that each piece can fit into the printer volume, size-wise. Herholz et al. [2015] decompose the surface of a 3D object into height field pieces, even allowing slight surface deformation for assurance, so that the pieces can be made into molds. While height field pieces are printing-friendly, their decomposition had a very different goal from ours, in that there was no consideration of using universal building blocks for the internal core.

Hu et al. [2014] decompose the solid volume of a given shape into a small number of approximately pyramidal parts. Our optimization problem also considers pyramidality, but it is only one of the criteria, for printing the residual; the core optimization involves a trade-off between the internal core and the residual. Moreover, our optimization computes not just one pyramidal decomposition, it must search over many pyramidal decompositions, each associated with a candidate solution in the iterative refinement process.

Block approximation. Approximating 3D shapes using a set of universal building blocks has been studied before, e.g., Legolization [Luo et al. 2015]. For fabrication, some recent works have explored the possibility of mixing 3D printing (for high-fidelity object parts) with block approximation (for lower-fidelity parts), where the latter is realized by LEGO brick assemblies in faBrickation [Mueller et al. 2014] or by 2D laser cutting [Beyer et al. 2015]. While faBrickation can speed up fabrication since assembling off-the-shelf bricks is faster than 3D printing, the result produced still leaves the low-fidelity boxy assemblies exposed — it does not reproduce the input 3D object. Moreover, the decomposition into 3D-printed parts and assembled parts in faBrickation [Mueller et al. 2014] was achieved via manual effort. In contrast, our method reproduces the input shape *as is*. More importantly, we develop a computational method to iteratively modify and optimize the building block assembly, as well as the decomposition of the residual shell, to reduce the overall fabrication cost via a coupled decomposition.

CofiFab. Song et al. [2016] build a 3D assembly of laser-cut panels to form an internal core inside a 3D-printed shell. While both our work and CofiFab fabricate an object by decomposing it into an internal core and a residual, there are several significant differences. First, CofiFab is *not* built from universal building blocks. The laser-cut panels, which form the internal core, are *custom-made* specifically for each target object. In our work, the internal core is assembled from universal pre-manufactured blocks regardless of the target shape. Second, CofiFab works most effectively for 3D objects whose interior can be approximated using a small number of convex polyhedra. Its cost increases as the object becomes more geometrically complex, as we show in Section 6. In our work, the building blocks are simple shapes whose assembly can flexibly adapt to various inputs regardless of their geometric complexity.

Computationally, CofiFab amounts to an approximate convex decomposition of an object's interior that minimizes the residual volumes. The residuals are not optimized for 3D printing. In our work, we pose and solve a novel problem of optimizing both the interior-residual split and pyramidal decomposition of the residual. Even though our problem is technically more challenging, our solution is efficient and leads to a reduction in the overall fabrication cost, while incurring far less support waste; see Table 2. Lastly, CofiFab is effective for fabricating larger objects due to the constraints pertaining to the size and thickness of the laser-cut panels to accommodate the mortise-and-tenon and halved joints. Typically, the smallest fabricated object presented in [Song et al. 2016] is ∼20cm high with 3mm plastic laser-cut panels. Our work can be naturally used for fabricating objects of varying sizes (large or small) and shapes by incorporating universal blocks in compatible scales.

Space tiling. This work is also related to tiling. Besides plane filling on a flat Euclidean plane, Eigensatz et al. [2010], Fu et al. [2010], and Singh and Schaefer [2010] independently developed methods to find a small set of shapes whose instances can be prefabricated and assembled to form a tiled surface that approximates a given input surface. Later, Zimmer et al. [2014] developed local mesh operators to iteratively refine a surface approximation of an input mesh covered by a set of simple 2D shapes. Our problem contains a tiling component for the internal core, which is related to solid tiling with space-filling polyhedra [Weisstein 2016]. However, the target volume is unknown in our case, and the tiling is a result of an optimization that considers the printability of the residual.

Volume approximation. Our method involves the construction of a "*filling*" (interior) volume inside a given 3D object, rather than a bounding (exterior) volume. Bounding volume approximation is a classical problem in computer graphics, especially collision detection [Ericson 2004]. Compared to previous works, one main distinction from our work is that we do not seek the best geometric approximation of the input object. Our approximation problem is

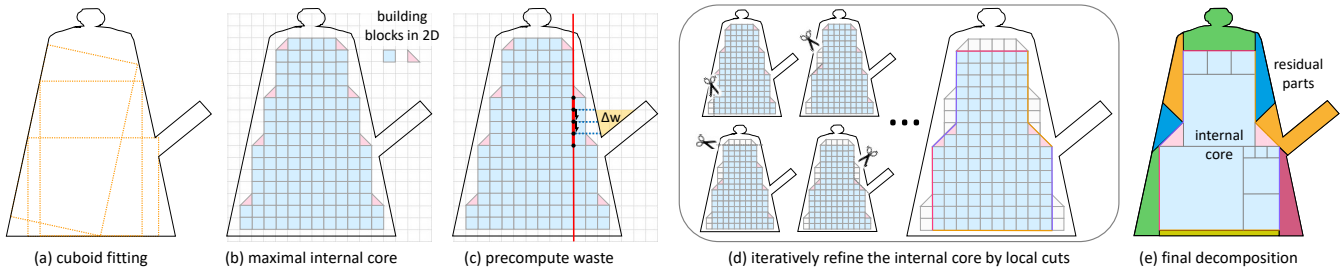


Fig. 3. Overview of our algorithm. It decomposes an input object shape (black contour in (a)) into (i) an internal core, which is an assembly of building blocks (b), and (ii) a residual, which is composed of approximately pyramidal parts, while reproducing the target object; see (e).

also novel in that it aims for a maximal internal volume while considering the decomposition and printability of the residual. While our goal here shares a similar spirit as [Jacobson 2017], we target to find a maximal internal core that composes of universal building blocks instead of being a replica of the given object itself. In terms of choices for the filling volume, k-DOPs, whose faces only take on one of k possible orientations, naturally fit the characteristics of our internal core as the assembly from our building blocks also has a limited number of face orientations. However, k-DOPs are convex but our internal cores need not be.

3 PROBLEM SETUP AND CHALLENGES

The input to our method is a user-provided 3D model in the form of a closed polygonal mesh. The output has two components: (i) an assembly of the universal blocking blocks for the internal core, and (ii) an enumeration of approximately pyramidal parts for the residual volume. The core computational problem of our method is that of a coupled decomposition, which targets for:

- *Faster fabrication speed*, by means of maximizing the size of the internal core, or equivalently, minimizing the residual volume, which requires online 3D printing;
- *Lower material cost*, by means of minimizing the residual volume and further decomposing it into printing-friendly pyramidal parts, which incur less support waste for FDM printing; and
- *Simpler assembly*, by means of reducing the number of residual parts in the pyramidal decomposition.

Clearly, these goals do conflict with one another. Maximizing the size of the internal core will overfit the input shape, resulting in a complex residual with a large number of residual parts. On the other hand, aiming for a simpler assembly expects fewer residual parts, or equivalently fewer cuts to partition the residual volume; however, we may then fail to produce residual parts with pyramidality. Furthermore, for aesthetics, we should avoid cutting salient regions over the object when partitioning the residual volume.

Objective function. To tradeoff among the goals, we formulate the following objective function to guide the decompositions:

$$\min\ (\ V + \alpha N + \beta W\ )\,, \tag{1}$$

where V is the residual volume, N is the number of residual parts, W is the amount of support (waste) material, and $\alpha, \beta \geq 0$ are the tradeoff parameters. Conceptually, when $\alpha, \beta \to 0$, we simply maximize the volume of the internal core. When $\alpha, \beta \to \infty$, it leads to a pure pyramidal decomposition with the fewest residual parts.

For simplicity and ease of illustration, we first present our method using 2D shapes, and then extend it to 3D at the end of Section 5.

Challenges. Since the assembly of the internal core from the building blocks follows a grid structure, as shown in Figure 3(b), a brute-force approach to seek the configuration that minimizes Eq. (1) would be to try different decompositions, as well as different grid orientations and positions. However, this would lead to a combinatorial explosion with intractable computation. Moreover, our computational problem involves two decompositions: the given object into internal core and residual, and the residual further into pyramidal parts. Considering them independently would lead to suboptimal solutions that cannot meet and balance the conflicting goals in the objective.

Furthermore, the objective function is clearly nonlinear, and with a tremendous search space, we must find an efficient search strategy. Last but not the least, the objective is also tedious to evaluate. Indeed, finding exact values of N and W necessitates a pyramidal decomposition, which is computationally expensive [Hu et al. 2014]. Therefore, we also need quick estimates for the terms in the objective function to support an efficient solution search.

4 OVERVIEW

As illustrated in Figure 3, our approach starts by fitting large cuboids inside the input object (a) to find a grid layout with which the object can enclose the largest number of the *cube* blocks. This leads to a maximal internal core as the initial decomposition (b). Moreover, potential support wastes aligned with the grid (c) are pre-computed. After that, we iteratively refine the decomposition using local cuts (d), while making use of the pre-computed waste information to quickly estimate cost terms in the objective function. After that, we decompose the residual into pyramidal parts by taking the internal core surfaces as the bases of the parts, while avoiding the salient surface regions marked by the user. Lastly, we assemble the 3D-printed residual parts with the internal core built from the universal building blocks to reproduce the final object (e).

A key philosophy behind our algorithm is that we start with the maximal internal core, which already minimizes the first term (V) in the objective. Hence, when we iteratively refine the internal core, as shown in Figure 3(d), we only need to focus on finding the best local cuts that help reduce N and W with a small sacrifice on V.

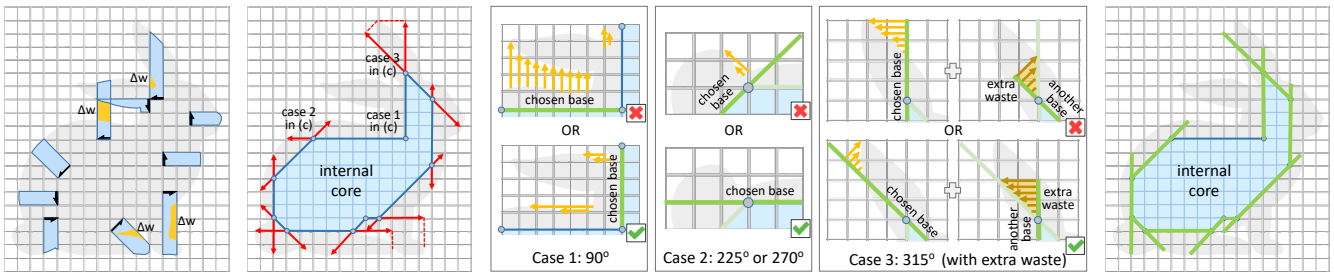


Fig. 4. (a) One-time precomputation of slab volume (blue & orange) and deficit volume Δw (orange) extended from each half edge. To estimate W for a given internal core, (b) we first find all the potential bases that can be taken to form residual parts; these include the boundary patches on the internal core (blue edges) and the partitioning lines extended from the corners between adjacent boundary patches (red arrows); (c) at each corner on the internal core (three major cases are shown above), we locally choose the base that incurs less support waste; note the green lines that indicate the chosen bases and the orange arrows that indicate the associated support wastes; and (d) we estimate W by summing up Δw over all the chosen bases (in green).

Other key ideas are applied to the iterative step, which is the most time-consuming part in our algorithm. This step involves three subproblems. The first is how to refine the internal core. For efficiency, we define a family of local cut operators to explore different ways of refining the internal core. These operators are simple and fast to construct, so we can quickly obtain a set of local refinement choices. The second subproblem is how to evaluate a refinement choice. For exact evaluation of the objective function, we need to compute a pyramidal decomposition of the associated residual in order to find N and W. This is, however, too costly even with a simplified pyramidal decomposition. Hence, we develop a fast approximate method to quickly estimate N and W with the help of the precomputed waste information. The last subproblem is how to explore the search space in a tractable manner. To this end, we formulate a beam search model that ranks the refinement choices and iteratively refines the internal core as guided by the objective function.

5 ALGORITHM

The major steps in our algorithm include the construction of a grid layout and maximal internal core, precomputation of support waste, iterative refinement of internal core, and pyramidal decomposition of the residual. We describe these steps in Sections 5.1- 5.4, which cover the 2D case, and present the 3D extensions in Section 5.5.

5.1 Grid layout and maximal internal core

To find the grid layout for forming the maximal internal core, we first construct a dense distance field inside the input object, and locate the interior points that are locally furthest away from the object surface. Centered at each of these points, we initialize a tiny cuboid with a random orientation, and then iteratively enlarge it and jitter its position and orientation to maximize its shortest distance from the object surface, while keeping it inside the object. By repeating this process, we generate M locally maximal cuboids inside the object (M=20), each defining a grid orientation; see Figure 3(a).

For each grid orientation, we create a grid layout and randomly jitter (translate) it in all directions, intending to maximize the number of full grid cells that can be enclosed in the object; see the blue cells in Figure 3(b) for an example. Note that the size of a grid cell equals the size of a square building block (cube, in the 3D case). Moreover, for printability, the residual should not be too thin, so we further require a minimum residual thickness ($d_{\min}$), and count only the full grid cells whose shortest distance from the object surface are larger than $d_{\min}$. In the end, we pick the grid layout that encloses the fullest grid cells and pack other types of building blocks around the full grid cells (with the $d_{\min}$ constraint) to form the maximal internal core; see the blue and red shapes in Figure 3(b).

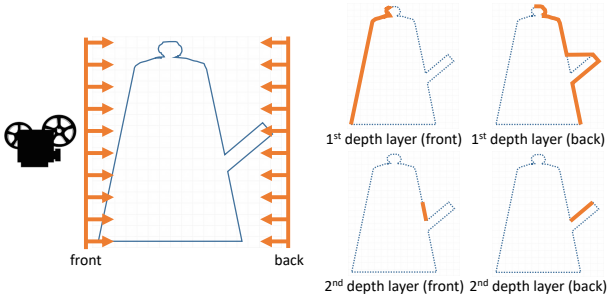


Fig. 5. Layered depth images produced from dual depth peeling for the view perpendicular to the vertical grid edges.

5.2 Precomputation of deficit (waste) volume

We perform a one-time pre-computation to support fast estimation of support waste needed in the subsequent iterative refinement step. This pre-computation first employs *dual depth peeling* [Bavoil and Myers 2008; Everitt 2001] to render layered depth images of the input object in specific views that are perpendicular to the grid edges and diagonals: vertical, horizontal, and two diagonals, in 2D; see Figure 5 for an example. By computing the depth difference between adjacent depth layers at a pixel, we can quickly determine an associated integral volume inside or outside the given object.

Next, we build a half-edge data structure over the grid layout. From each half edge, we extend it towards its right till the outermost object surface to form a slab volume, and determine its associated deficit volume (Δw), which is the subvolume inside the slab but outside the object; see the blue and orange regions in Figure 4(a)

for examples. Such deficit volume reveals the amount of support waste in case the associated half edge becomes a base of a pyramidal residual part. Thanks to the layered depth images, we can quickly determine the deficit volume by computing the depth difference across depth images in the view corresponding to the half edge.

Note that by computing the slabs via orthographic projections, we assume that exterior spaces directly under a printed layer need to be filled by support material. This is certainly an *over-estimate* of support waste, since in practice, FDM printers can build up printed layers along a *slope*. For our purpose of obtaining a fast approximate estimation of W, we are content with such a result.

5.3 Iterative refinement of internal core

Before we present the iterative refinement procedure, we first explain how we estimate N and W in the objective function, and how we use local cuts to refine an internal core.

(i) Estimation of N and W. To estimate the residual part count (N) for a given internal core, we group connected co-planar half edges (half faces, in the 3D case) on the internal core surface into planar boundary patches, e.g., the internal core shown in Figure 3(e) has eight such patches. Since most boundary patches will become bases of pyramidal residual parts, we estimate N as the patch count.

Before we present our procedure to estimate W, we first discuss the key idea behind it. Taking "case 1" marked in Figure 4(b) (correspondingly, see case 1 in Figure 4(c)) as an example, we have a pair of adjacent boundary patches next to a 90° corner on the internal core. In this case, we may take *either* boundary patch, the vertical or the horizontal patch, as the base for forming a 3D-printed residual part. To choose between them, we can estimate the amount of support waste that will incur for each choice, and take the one with less waste. Similarly, for the other cases (corners) shown in Figure 4 (b & c), we also have choices of bases for forming residual parts. By summing up the support waste over all the chosen bases, we can then find an estimate of the overall support waste W.

There are three substeps to estimate W. In the first substep, we find all potential bases for forming residual parts, and for each base, estimate its associated support waste. In detail, there are two kinds of base: (i) boundary patches on the internal core, and (ii) partitioning lines (planes, in the 3D cases) extended in parallel from the boundary patches (until reaching the object surface) for dividing the residual volume into parts. We do not consider arbitrary partitioning lines, since extending partitioning lines from boundary patches keeps the residual part count (N) and leads to a simple and efficient residual decomposition. In the end, we sum Δw over the half edges on each base to estimate the support waste of each base. Note that for each partitioning line, we locate the connected object surface above it and project the surface onto its plane; if the projected image goes beyond the partitioning line, we extend the partitioning line further to cover the projected image for a more accurate estimation of the support waste; see the dashed red lines in Figure 4(b) for examples.

The second substep iterates the corners on the internal core and chooses the bases that incur less waste. There are three major cases in this substep (see also Figure 4 (b & c)). The first case compares potential bases at 90° corners, while the other two compare potential bases (partitioning line pairs) at 225°/270°, and 315° corners:

- For the 90° corner case (see "case 1" in Figure 4(c)), we choose between the two adjacent boundary patches. In the example shown in the figure, we choose the vertical base (bottom choice), since it incurs less waste than the other choice.
- For the 225°/270° case (see "case 2" in Figure 4(c)), we choose the base on the bottom (which includes a boundary patch and its extended partitioning line), since it incurs zero waste.
- Besides the regular support waste, the third case (see "case 3" in Figure 4(c)) shows situations with an extra waste along a slope. To illustrate the extra waste, we take the bottom choice in this case as an example, where the chosen base forms the residual part that covers the top-left portion of the ear of the rabbit. Next to this residual part, we have another residual part for the lower-right portion of the ear; since the ear portion here will be printed at a slope over "another" base, it thus incurs an extra waste; see the figure for the illustration.
- Note also that we ignore the case for 45° corners, since they are rare. Out of the sixteen 3D shapes we tested in the experiments, only two of them have a few 45° corners, and these corners were found to incur zero waste. On the other hand, we ignore the case for 135° corners for efficiency concern. To handle 135° corners, we have to compute the geometry of each base and calculate the waste separately to avoid summing duplicated waste, which is computationally very tedious.

Lastly, the third substep sums up the support wastes (Δw) over all the chosen bases to find the estimated W; see Figure 4(d). It is important to note that we design this approximate method with efficiency in mind. Taking the Horse model shown in Figure 23 as an example, our method only took 35∼77 milliseconds to estimate W (which is crucial to support the iterative refinement) as compared to ∼17 min. for a general pyramidal decomposition [Hu et al. 2014] and ∼12.6 sec. for the simplified pyramid decomposition to be presented in Section 5.4. Moreover, an experiment to be presented in Section 6 also shows that the trend of W estimated by our method roughly follows the trend of W estimated by Cura [Ultimaker ltd. 2017]. Lastly, for shapes with extensive overhangs, the estimated W may have high errors; see the limitations in Section 7 for the details.

(ii) Local cut operators. As an initial solution, the maximal internal core usually overfits the object interior with an irregular boundary; see Figures 6 and 7. Hence, we will employ the objective function as a guidance to iteratively refine the internal core to reduce N and W with least increase in V. This means that to optimize the refinement, each step should not bring a large increase in V, but at the same time, should simplify the internal core for smaller N (fewer boundary patches) and desirably smaller W (less support waste). Additionally, the refinement should be simple to construct and fast to compute, since we will need to explore many refinement choices in the search process. With these criteria in mind, we design the following local cut operators to produce candidates in each refinement step:

- *Boundary cut* takes a boundary patch on the internal core as a cut plane to remove a small connected component (locally next to the patch) from the internal core; see Figure 6(a);

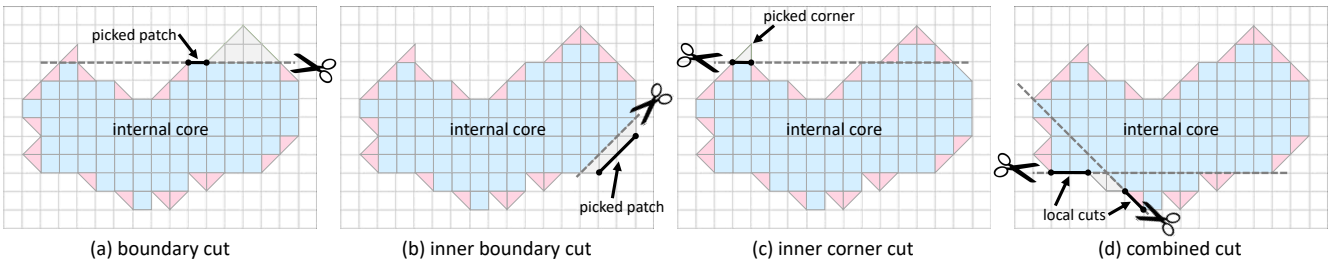


Fig. 6. The four local cut operators (a-d) considered in our solution for refining the internal core and improving the objective function.

- *Inner boundary cut* removes a slightly larger connected component for efficiency concern by moving the cut plane in the case of boundary cuts one unit deeper; see Figure 6(b);
- *Inner corner cut* removes a small connected component around a sharp corner with a cut plane one unit below; see Figure 6(c); and
- *Combined cut* removes a tiny component in a concave region on the internal core, thereby leading to a high chance of reducing N. To form a combined cut, we find a pair of nearby local cuts formed by some other operators, and take the intersection volume of their associated connected components as the region to be removed by the combined cut operator; see Figure 6(d).

(iii) Iterative refinement. Recalling the problem setting described in Section 3, we start with a maximal internal core with minimum V, and formulate a beam search model to iteratively refine the internal core (denoted as P) with local cuts and improve the objective function in each iteration; see the procedure outlined below.

K = beam search width
P = construct an initial decomposition (maximal internal core)
P^* = P // initialize the optimal decomposition
P_K = { P } // initialize the set of K best decompositions with P
while true **do**
  S = find a set of local cut candidates on P_K
  S_K = pick K cuts in S with best positive gain in Eq. (1)
  if S_K is not empty **then**
    P_K = apply S_K to update K best decompositions
    P^* = best decomposition in P_K
  else if α < upper bound of α **then**
    α += $\delta\alpha$
    continue
  else
    break
  end if
end while

For a given internal core, we obtain a set of local cut candidates by applying a boundary cut and an inner boundary cut for each boundary patch, and an inner corner cut at each sharp corner. Moreover, we take a pair of nearby cuts to form a combined cut, if their local-connected components have a nonempty intersection volume. In practice, this may lead to several thousand candidates for a 3D internal core, so we filter out those that remove more than 10% of the internal core volume. For each candidate, we estimate its N and W and deduce its gain in the objective function, i.e., $\Delta V+\alpha\Delta N+\beta\Delta W$ (see Eq. (1)), where ΔV, ΔN, and ΔW are the amount of reductions in V, N, and W, respectively. In each iteration of the beam search model, we find the K local cuts that lead to the best positive gains, and apply them to refine the internal cores in P_K accordingly.

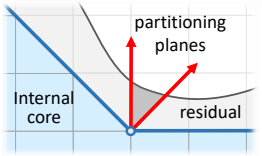
The initial decomposition is basically a solution for which we optimize the objective function with α=β=0, since the maximal internal core simply minimizes V. Empirically, we found that if we set α to its respective target value when the iterative procedure starts, we may easily trap at some local minima too early in the search process. Hence, we initialize α to zero and gradually increase it by $\delta\alpha$, every time we cannot find any local cut candidate with positive gain in S_K. This strategy is similar in spirit to the weight decay concept [Hertz et al. 1991]; essentially, when we gradually increase α in our case, we gradually reduce the influence of V.

5.4 Final decomposition

After obtaining the optimized internal core, we have two final substeps, one on the internal core and the other on the residual.

For the internal core, we greedily arrange first the larger-size building blocks, which, as illustrated in Figure 3(e), can be fabricated either by pre-printing (or molding) or by pre-assembly of smaller blocks, so that we can assemble the internal core from fewer parts.

For the residual, we further decompose it into parts using a *simplified* pyramidal decomposition, which takes the internal core surfaces (boundary patches) as bases rather than arbitrary partitioning planes as in the general pyramidal decomposition [Hu et al. 2014]. In our implementation, we first find all the potential partitioning planes, including the extended partitioning planes we identified earlier, e.g., see the red arrows in Figure 4(b), as well as additional partitioning planes perpendicular to the bases specifically for the case of 135° corners; see the inset figure. Then, we apply them to over-segment the residual volume. Now, we should have the full geometry of each residual fragment for more accurate estimation of support waste, so we merge fragments into residual parts by choosing partitioning planes that incur less support waste. This is a binary decision by checking possible merges at every corner (or edge, in 3D) on the 2D internal core.



Salient regions. Users may also mark salient regions on the input shape, so that the binary decision can avoid cutting through salient

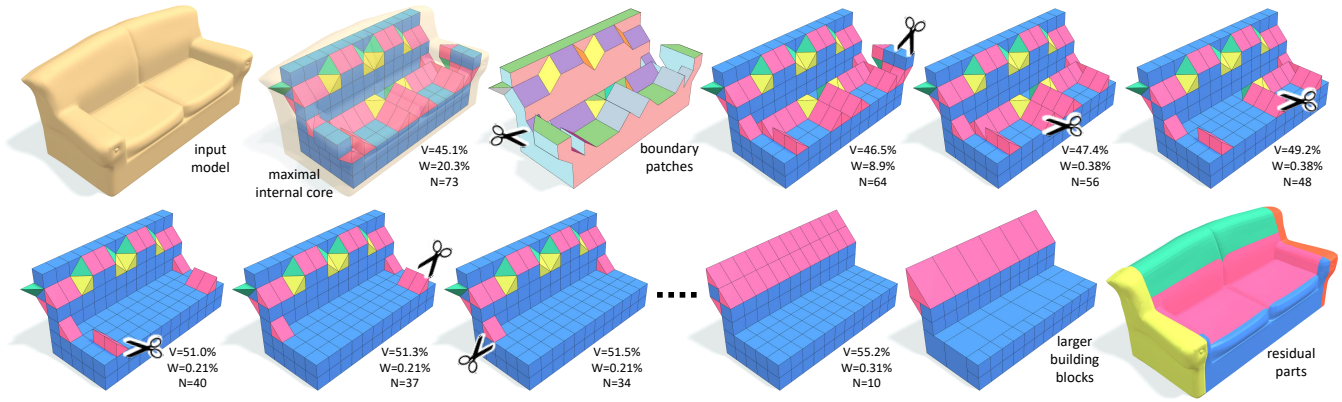


Fig. 7. A running example showing our algorithm at work on the Couch model in 3D; see the changes in the estimated V, W and N values over the iterations. Note that V and W are presented in proportion (%) to the whole object volume.

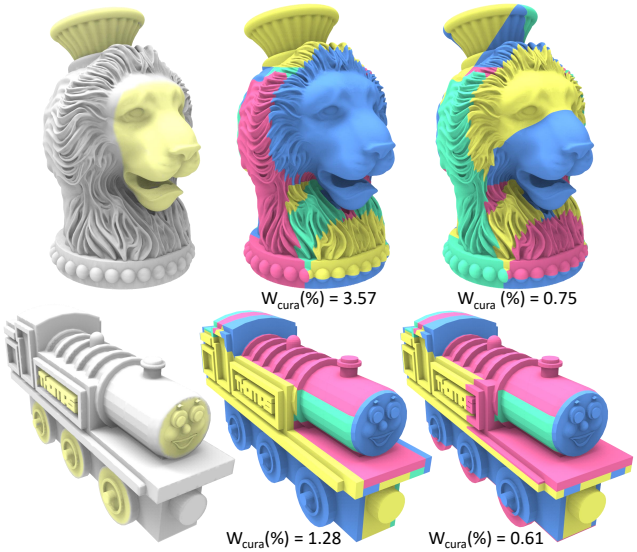


Fig. 8. From left to right, input shapes with user-marked salient regions (in yellow), pyramidal residual decomposition with and without avoiding salient regions; W_{cura} denotes the W estimated by Cura. It is presented in proportion (%) to the whole object volume.

regions; see Figure 8 for results. This may lead to an increase in support waste (W), but N and V will remain unchanged.

5.5 Extension to 3D

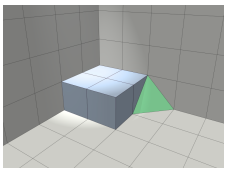
We now extend the algorithm described so far to 3D with the following amendments. First, we employ four different shapes as the universal building block set; see Figure 1. Moreover, we use a 3D grid layout and form a 3D maximal internal core; see Figure 7.

Second, when we use the dual depth peeling method to generate the layered depth images, we consider thirteen orthographic views in 3D, and extend the half-edge data structure to half-face data structure in the 3D grid to store the precomputed deficit volume.

Recalling that a boundary patch is a group of connected co-planar faces (half faces) on the internal core, their topology in 3D is more complex than that in 2D; see the third image on top of Figure 7. Hence, we build a graph data structure to encode their connections: a node for each patch and an edge for each connection. Using this graph, we can then find local cut pairs that are topologically close to each other for forming the combined cut candidates.

Fourth, to estimate W in 3D, potential bases include the boundary patches on the internal core and partitioning planes extended from sharp edges (225°, 270° & 315°) shared between boundary patches.

Combined cuts with two cut planes may not be sufficient in 3D. Thus, we introduce the *cavity cut* to remove a small connected component at the corner met by three boundary patches at angles $\leq 135°$; see the inset figure for an example.



Lastly, we make binary decisions at edges instead of at corners between adjacent boundary patches to determine partitioning planes in a 3D pyramidal decomposition. Unlike the 2D case, we may have unassigned residual fragments, e.g., at exterior corners, where three boundary patches meet. Hence, we use the same principle, i.e., less support waste, to assign (i.e., attach) these fragments to nearby residual parts to complete the pyramidal decomposition.

6 RESULTS AND EVALUATION

In this section, we present results produced from our algorithm on various 3D shapes, and present experiments conducted for evaluating different aspects of our algorithm.

Parameters. Our algorithm has the following parameters: weights in the objective function ($\delta\alpha$, α, and β), branching factor in beam search (K), minimum residual thickness ($d_{\min}$), and size of grid cells or building blocks (l). All the results shown in the paper are generated using $\delta\alpha$=1, α=2, β=1, K=5, and $d_{\min}$=3mm. Note that the value of l depends on the size of the building blocks we employed.

Decomposition results. Figure 23 shows a gallery of our results: the internal cores are assembled from the universal building blocks, while the residuals are composed of approximate pyramidal parts.

Table 1. Comparing our method to conventional 3D printing. Notations: residual volume (V); residual part count ($N_{res.}$); waste volume (W); building block count (N_{block}), which includes blocks in finest (l=2cm) and double (l=4cm) scales; time to run our method ($T_{algo.}$) in min.; amount of material (M) in gram; volume of support material ($V_{sup.}$); time to print and assemble residual parts (T_{total}) in hours; costs (C) in USD; and time to fabricate the whole object ($T_{obj.}$) in hours. In the 1st column, asterisks indicate models with salient region mark-up, while the last two columns show the overall percentage savings of our method in terms of fabrication cost and time. Note also that the Cura Software [Ultimaker ltd. 2017] is used to estimate the values of M, $V_{sup.}$, and 3D printing times.

	Results of Our Method					Estimated 3D Printing with Our Method								Estimated Conventional 3D printing						% Reduction	
model	V (%)	$N_{res.}$	W (%)	N_{block}	$T_{algo.}$ (min)	$M_{res.}$ (g)	$M_{sup.}$ (g)	M_{total} (g)	$V_{sup.}$ (%)	T_{total} (h)	$C_{res.}$	C_{block}	C_{total}	$M_{obj.}$ (g)	$M_{sup.}$ (g)	M_{total} (g)	$V_{sup.}$ (%)	$T_{obj.}$ (h)	$C_{obj.}$	$C_{reduc.}$	$T_{reduc.}$
Arch	32.1	16	0.05	300	46.7	2162	45	2207	0.79	187.8	44.1	11.4	55.5	5700	636	6336	11.16	514.4	126.7	56.2	63.5
Bimba	62.4	25	0.35	64	22.7	644	7	651	0.82	58.5	13.0	3.1	16.1	849	278	1127	32.74	93.7	22.5	28.5	37.5
Couch	55.2	9	0.31	78	12.8	1223	3	1226	0.17	104.4	24.5	4.0	28.6	1808	325	2133	17.98	173.2	42.7	33.1	39.7
Double torus	60.7	24	0.60	77	6.5	2003	10	2013	0.39	177.7	40.3	5.2	45.4	2577	736	3313	28.56	270.0	66.3	31.4	34.2
Duck	62.0	18	0.60	63	19.5	1202	36	1238	2.11	106.1	24.8	4.0	28.8	1706	327	2033	19.17	166.1	40.7	29.2	36.1
Fandisk	52.4	14	3.80	79	6.2	725	48	773	4.20	63.2	15.5	3.4	18.8	1143	47	1190	4.11	96.5	23.8	20.8	34.5
House	34.3	7	0.71	28	2.6	572	9	581	0.65	49.1	11.6	4.0	15.7	1376	383	1759	27.83	141.6	35.2	55.5	65.3
Letters SIG	45.9	34	2.19	63	4.2	961	5	966	0.31	85.6	19.3	5.4	24.7	1637	0	1637	0.00	130.9	32.7	24.5	34.6
Ns	32.1	24	0.11	240	35.9	3655	0	3655	0.00	286.2	73.1	33.8	106.9	8870	2034	10904	22.93	874.1	218.1	51.0	81.5
Trophy	59.3	23	0.06	52	7.0	564	1	565	0.14	52.1	11.3	2.1	13.4	704	297	1001	42.19	82.1	20.0	32.9	36.5
*Horse	65.9	21	9.90	84	41.5	1274	226	1500	10.21	138.1	30.0	3.7	33.7	2214	1586	3800	71.64	314.5	76.0	55.7	56.1
*Sphynx	54.4	13	1.42	58	5.1	452	8	460	1.17	41.2	9.2	1.5	10.7	682	45	727	6.60	60.1	14.5	26.2	31.4
*Squirrel	54.9	20	0.06	80	43.5	745	0	745	0.00	68.8	14.9	3.8	18.7	1053	269	1322	25.55	108.8	26.4	29.4	36.7
*Toy train	46.8	16	0.32	91	78.4	1829	38	1867	1.28	162.1	37.3	5.1	42.5	2963	1361	4324	45.93	349.7	86.5	50.9	53.7
*Triceratops	62.7	20	5.75	102	33.3	1132	126	1258	8.74	116.6	25.2	5.7	30.9	1441	907	2348	62.94	193.2	47.0	34.3	39.6
*Vase-lion	57.8	22	1.98	53	15.1	568	28	596	3.57	56.0	11.9	3.2	15.1	785	216	1001	27.52	84.5	20.0	24.5	33.7

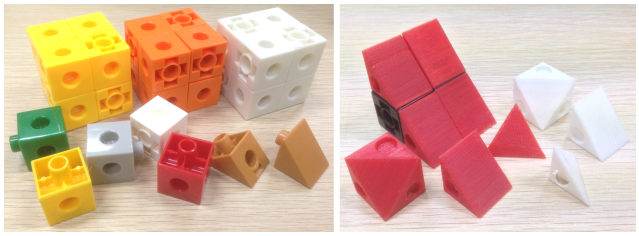


Fig. 9. Building blocks employed in this work. Left: from a consumer brick set. Right: blocks from a low-end 3D printer (in red) and blocks from a higher-end 3D printer (in white).

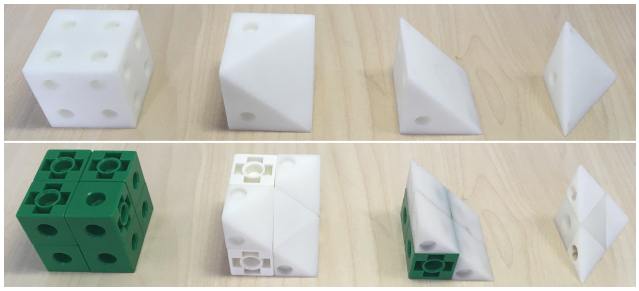


Fig. 10. Larger building blocks made by pre-printing (top) or by pre-assembly from smaller made building blocks (bottom).

We tested a rich variety of models, from house and furniture, to animals, toys, and statue, etc. The model shapes have varying degrees of complexity, and some have complex non-bulky interiors. Next to each figure pair, we show the model name and the associated percentage reduction in the estimated fabrication time and cost as compared to conventional 3D printing. These results show that our method brings positive improvements for all the presented models. Note also that the shapes of the internal cores optimized by our method; these shapes not only abstract the interior of each model but are also optimized for the benefit of using the universal building blocks, as well as the printability of the residual parts.

Preparing the building block set. Our primary choice of the building blocks is a consumer brick set, which provides cubes (l=2cm) and triangular prisms; see Figure 9 (left). The cost of each cube is only around US$0.012. Moreover, we may *pre*-print the building blocks, particularly for the non-cube types, since the consumer brick set provides mainly cubes with limited triangular prisms. For 3D printing with PLA material, the cost is around US$0.06 per cube, given PLA material at US$0.02 per gram. We tried both Ultimaker 2+ Extended (a higher-end 3D printer) and TierTime UP Plus 2 (a low-end 3D printer) to produce the 3D-prints; see Figure 9 (right). In addition, we explored molding as a means to create the building blocks; a quoted cost we got in making a mold is US$145 and that in making a cube block (l=2cm) is around US$0.058 for an order of ten thousand pieces. Lastly, for larger building blocks, we may *pre*-assemble them from smaller building blocks, or *pre*-print them to make the internal volume hollower; see Figure 10 for examples.

Statistics of results and comparison. Table 1 shows the statistics of results in four parts. From left to right, the first part shows the basic results of our method: V and W are in proportion (%) to the whole object volume, $N_{res.}$ is the residual part count, N_{block} is the building block count, and $T_{algo.}$ is the time taken to run our method on a desktop computer with Intel i7-6700K 4GHz CPU and 16GB RAM. Overall, we can see that our method produces large internal cores with not too many residual parts. Particularly, it finishes its computation in the order of minutes, which is negligible as compared to the time needed to fabricate and assemble the residual (T_{total}).

The second and third parts in Table 1 compare the material consumption (M), fabrication time (T) and cost (C) between our method and conventional 3D printing. The T, M, and support volume ($V_{sup.}$) quantities in these two parts are estimated using Cura Software (Version 15.04 [Ultimaker ltd. 2017], which is a 3D-print preparation software, with the following settings: using the 3D printer Ultimaker 2+ Extended and a filling rate of 43.85%. In detail, T_{total} includes the time taken to print our method's residual parts and to attach them onto the interior core (note that we assemble the

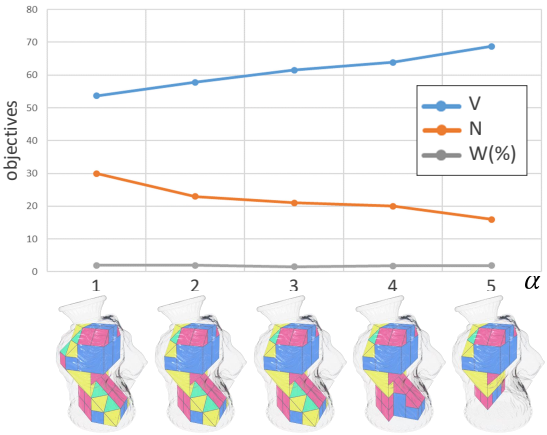


Fig. 11. Effect of parameter α on V, N and W.

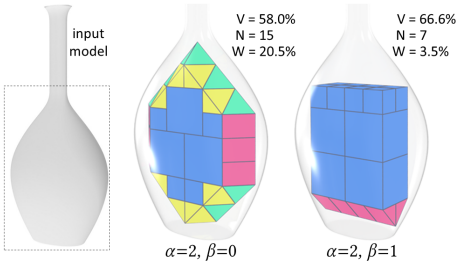


Fig. 12. Effect of parameter β, particularly on W.

interior core from building blocks while the residual parts are being printed); empirically, the time taken by a single student to attach the residual parts onto the interior core is around 20 to 30 min.; $T_{obj.}$ is the time taken to fabricate the whole object in the case of conventional 3D printing, $V_{sup.}$ is the associated support volume, while $M_{res.}$, $M_{obj.}$, and $M_{sup.}$ are the amount of print material for the residual parts (our method), the whole object (conventional), and the associated supports, respectively. Hence, $M_{total} = M_{res.} + M_{sup.}$ for our method, and $M_{total} = M_{obj.} + M_{sup.}$ for conventional 3D printing.

The cost (C) terms in Table 1 are estimated based on (i) the price of a cube block in the consumer brick set (US$0.012); and (ii) the PLA material cost (US$0.02 per gram). Using these numbers, we can estimate the cost of printing the residual parts (our method) and printing the whole object (conventional), i.e., $C_{res.}$ and $C_{obj.}$. To estimate the cost of building blocks C_{block}, we multiply US$0.012 with the number of finest scale building blocks instead of N_{block}, since N_{block} considers larger building blocks. Hence, the total cost of our method C_{total} is $C_{res.} + C_{block}$.

From the above numbers, we can deduce how much our method saves in terms of the overall fabrication time and cost as compared to conventional 3D printing; see the last part in Table 1. Comparing C_{total} and $C_{obj.}$, we show that our method saves ~36.5% of fabrication cost on average; the saving is not only in terms of print material by means of using low-cost building blocks to form the object interior, but also in terms of the reduction in support material by considering pyramidal decomposition of residual early in our

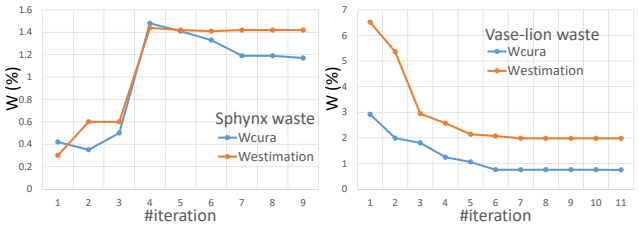


Fig. 13. Plotting our estimated waste (in orange) vs Cura estimates (in blue) for the Vase-lion and Sphynx models over all iterations.

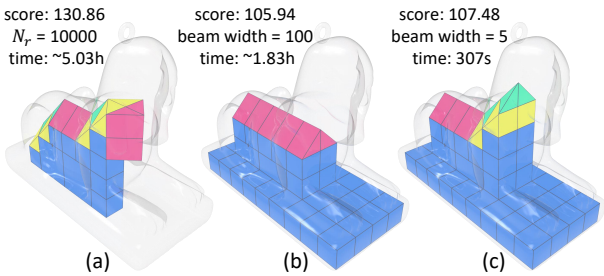


Fig. 14. Our solutions found for the Sphynx model using (a): random search, (b): beam search with width = 100, and (c): current solution.

computation. Comparing T_{total} and $T_{obj.}$, our method saves ~44.7% of fabrication time on average, since the time taken to compute our method is negligible as compared to the 3D printing time.

Effects of parameters in objective function. A larger α puts more emphasis on minimizing N, so the beam search process will try to find decompositions with fewer residual parts. However, residual volume (V) will increase, as a tradeoff. Figure 11 shows an example on how α affects the decomposition results of Vase-lion, especially its effect on V and N. The other parameter in the objective function is β, which associates with the waste term. From the results shown in Figure 12, we can see that if β is set to zero, meaning that we do not care the amount of waste in the result; the estimated waste can then rise up to as high as 20% of the object volume. This result shows that our optimization model is able to avoid decomposition solutions that lead to a large amount of support waste.

Waste estimate for W. Figure 13 shows, for two models, the result of our estimated support waste vs. estimates provided by Cura, which we treat as the ground truth. As can be observed, over all solution iterations, the two estimates exhibit similar trends. Test results on other models also show that the trend seen in Figure 13 is quite representative; hence, it presents a strong indication that our W estimate holds merit when applied to rate the decompositions.

Optimality assessment. To explore how close our current solution is to being the global optimum, we make the solution search significantly more exhaustive and compare the solutions found to our current one. Since finding optimal solutions analytically, even for simple examples, is highly unlikely, we hope that by making the search highly exhaustive, we can obtain solutions that are close to being optimal. To this end, we tested two exhaustive search options: random search and beam search with a much larger beam width

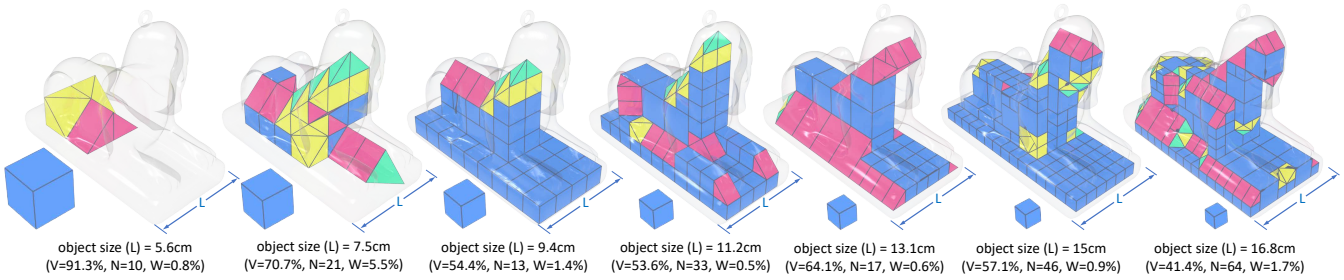


Fig. 15. Our results for the Sphynx model in varying scales. We report the object size (i.e., the model width L), residual volume $V(\%)$, residual part count N(estimated), and support waste $W(\%)$ below each result, and show a unit cube of fixed size (2cm) on the lower left of each result to reveal the scale.

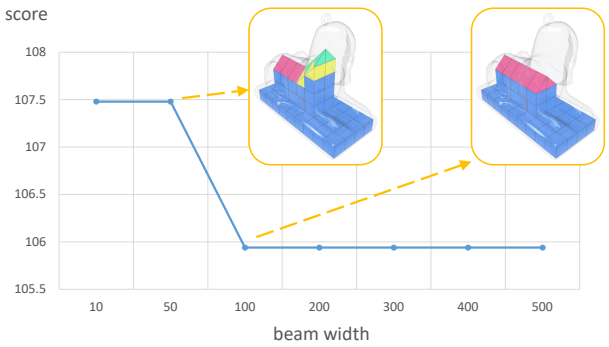


Fig. 16. Plot of objective function values over beam width.

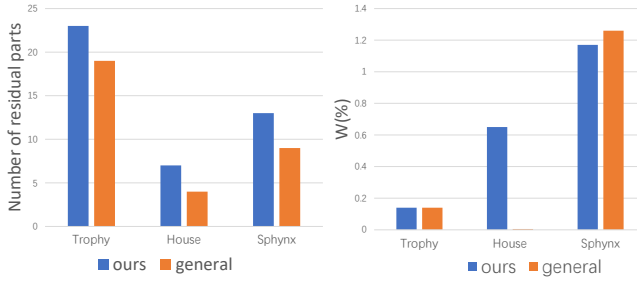


Fig. 17. Simplified pyramidal decomposition (denoted as ours) *vs.* general pyramidal decomposition (denoted as general).

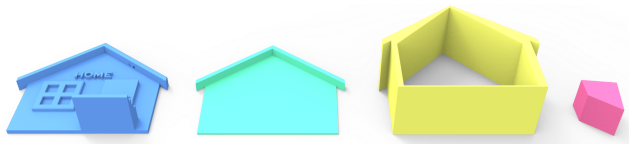


Fig. 18. The optimal pyramidal decomposition created by a participant on the House model has only four residual parts and nearly zero support waste.

(K) than our current choice. The random search starts with the maximum internal core, then we randomly select a child from all possible cuts, and repeat until we empty the internal core. We repeat the random search N_r=10,000 times and pick the best solution. For the large-width beam search, we keep α and β constant throughout, and tested K with 10, 50, 100, 200, 300, 400, and 500.

Figure 14 shows the solutions found on the Sphynx model with the three options, including the solution using our default setup (e.g., beam width (K) is set to 5). As one would expect, our current solution outperforms random search and is bested by a beam search with width 100; our current solution, however, only takes minutes to compute and yet its result is already comparable to a beam search with much larger width. Figure 16 shows that, by varying the beam width, the objective function value quickly stabilizes and does not change after the width exceeds 100, while solutions obtained for beam width 5 or 10 are not so far off.

Comparison to general pyramidal decomposition. We also explored the optimality of our simplified pyramidal decomposition in terms of residual part count and support waste amount as compared to a *general pyramidal decomposition*, where the partitioning planes can be arranged arbitrarily in 3D. Taking the residual volumes of the Trophy, House, and Sphynx models as inputs, we recruited three researchers who are experienced in geometry processing to help divide the residual volumes into minimal approximate pyramidal parts. Specifically, the residual volumes were presented as 3D meshes in Maya and were cut into parts using Boolean operators; overall, each user took around 5 to 10 min. to come up with a decomposition solution and another 20 to 30 min. to cut the residual. In the end, for each model, we pick the decomposition with the lowest residual part count as the general pyramidal decomposition result, and then estimate the support waste associated with the residual parts.

Figure 17 shows the comparison results. On the left, we can see that general pyramidal decomposition usually leads to fewer residual parts, since it allows arbitrary partitioning planes. On the right, we can see that our method has comparable support waste for two of the three models, except for the House model. By cleverly arranging the partitioning planes for House (see Figure 18), we can even obtain an optimal decomposition with nearly zero support waste. More visual comparisons can be found in the supplemental material.

Our results in varying scales. Taking the Sphynx model as an input, we linearly scale it and generate results under different scales. Figure 15 presents the results, showing that our algorithm can work for every scale, while keeping the same size for the universal blocks.

Comparison to CofiFab. We obtained the code of CofiFab from the authors and ran it on several models. For quantitative comparison, we also ran our method on the same set. Table 2 summarizes the

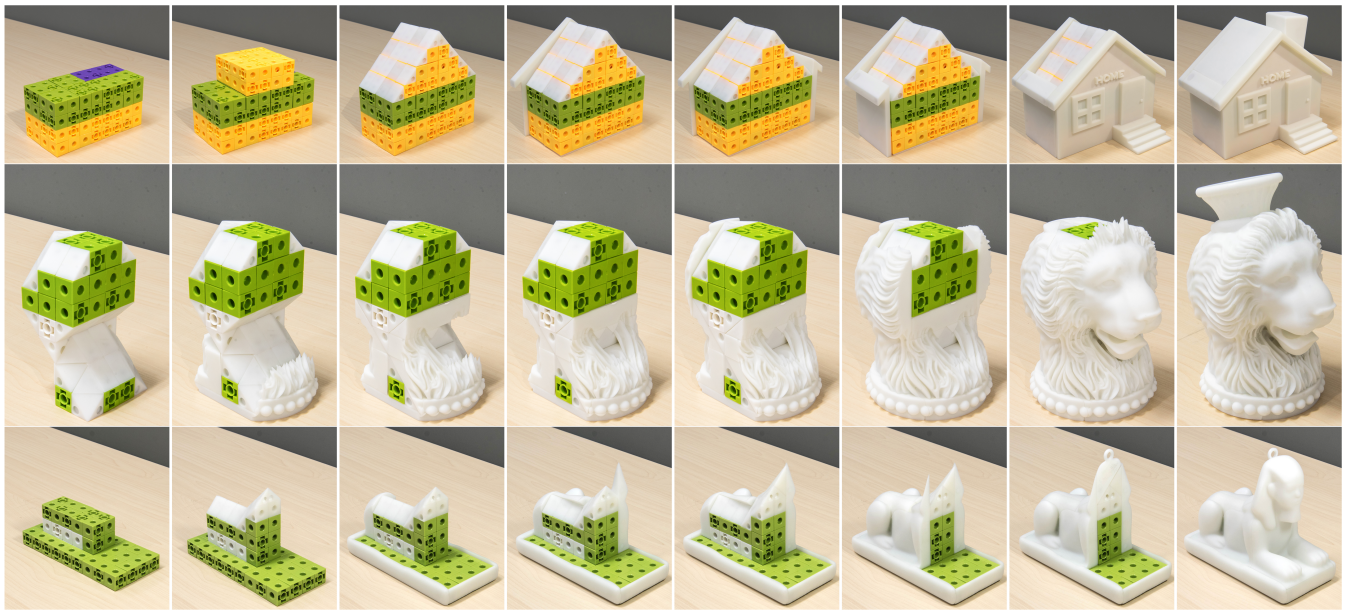


Fig. 19. Physical fabrication results. Here we show the assembly sequences of the House (top), Vase-lion (middle), and Sphynx (bottom) models.

Table 2. Comparing the cost-efficiency of our method and CofiFab.

Ours *vs.* CofiFab		N_{panel}		$N_{res.}$		$M_{sup.}$ (g)		V (%)		C_{total}	
		Ours	Cof.	Ours	Cof.	Ours	Cof.	Ours	Cof.	Ours	Cof.
Bulky shape	Bimba	-	21	25	20	7	19	62.4	47.3	16.1	20.5
	House	-	7	7	7	9	11	34.3	30.5	15.7	21.3
	Squirrel	-	30	20	24	0	83	54.9	52.3	18.7	24.7
	Vase-lion	-	24	22	22	28	60	57.8	50.4	15.1	21.9
Complex shape	Arch	-	38	16	34	45	143	32.1	37.7	55.5	76.5
	Donut	-	36	18	28	0	63	60.0	58.2	28.7	34.2
	Duck	-	36	18	32	36	106	62.0	62.5	28.8	38.1
	Ns	-	48	24	40	0	202	32.1	34.9	106.9	101.3
	Toy train	-	29	16	25	38	89	46.8	40.5	42.5	53.5

results: N_{panel} denotes the number of laser-cut panels for building CofiFab's internal cores, $N_{res.}$ denotes the number of residual parts, $M_{sup.}$ denotes the amount of Cura-estimated support wastes, V denotes the residual volume, and C_{total} denotes the overall estimated fabrication cost in USD. Note that to estimate the fabrication cost of CofiFab, we consider a common laser-cut panel material, which is plastic (acrylic) with 3mm thickness and US$10.8 for 80cm×80cm. Besides tabular comparison, we provide side-by-side comparisons of two sets of decomposition results in the supplemental material.

From Table 2, we can see that for complex shapes, CofiFab requires more laser-cut panels (N_{panel}) to create more convex polyhedra, so that the interiors of the complex shapes can be more fully filled by the internal cores formed by the panel assemblies. Since CofiFab creates one residual part roughly for each internal core face, it thus produces more residual parts ($N_{res.}$) over the exterior of the fabricated object as compared to our method. Moreover, CofiFab incurs more support waste ($M_{sup.}$) than our method in the 3D printing of the residuals parts, since our method jointly considers the residual part count and waste material early in the optimization. Furthermore, we can also see from Table 2 that the overall fabrication cost (C_{total}) of our method is usually lower than that of CofiFab, since we maximize the use of low-cost building blocks and minimize the support waste incurred in printing the residual parts.

Last but not least, we would highlight that all parts, in both residual and internal core, fabricated by CofiFab are *custom-made*. Hence, it has relatively less restriction on the parts geometry; by customizing more laser-cut panels and creating more convex polyhedra, it can assemble an internal core that better fits the object interior with a smaller residual volume (V), e.g., the internal core in CofiFab's Arch model only has six convex polyhedra and 38 laser-cut panels. In contrast, we aim to maximize the benefit of using *universal* building blocks in 3D fabrication; the same set of building blocks is used to form the internal core for various objects. Interestingly, from the results shown in Table 2, we can see that by using our universal building blocks of just four types, our method can still achieve residual volumes (V) whose sizes are mostly comparable to those of CofiFab. Additionally, our method involves fewer residual parts, less support waste, and lower overall fabrication cost.

Physical fabrication. Figure 19 shows physical fabrication results of the House, Lion, and Sphynx models. When producing these models, we attach the 3D-printed residual parts onto internal cores using blu-tack, which is a reusable adhesive, allowing us to disassemble and re-assemble the structure. Also, when computing the volume decompositions, we consider tolerance between the parts.

Extension with LEGO-built internal core. We can easily extend our method to allow the use of LEGO bricks to build an entire internal core in the form of a regular voxelized shape; see Figure 20 (left). We only need to slightly modify our algorithm by having only cubes in our building block set (but with side length 1.58cm, according to the

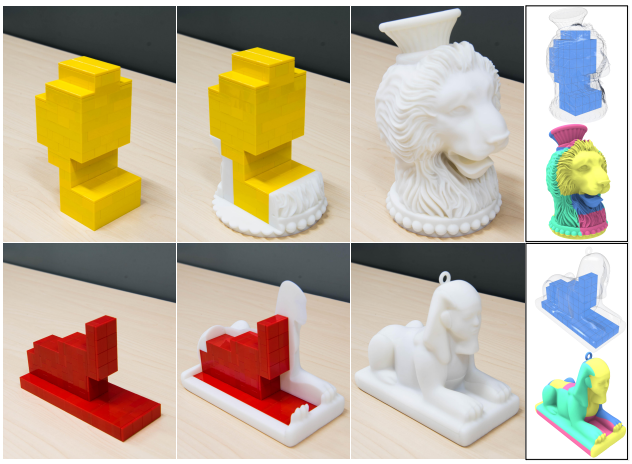


Fig. 20. Physical fabrication results with LEGO-built internal core. From left to right: internal cores built from LEGO bricks, some residual parts attached, the completed assemblies, and then the internal core and the parts composition in virtual.

Table 3. Quantitative comparison with LEGO-built internal core.

	Results			Estimated 3D Printing					
model	V (%)	$N_{res.}$	W (%)	$M_{res.}$ (g)	$M_{sup.}$ (g)	M_{total} (g)	$V_{sup.}$ (%)	T_{total} (h)	$C_{res.}$
Sphynx	54.4	13	1.4	452	8	460	1.2	41.2	9.2
Sphynx (LEGO)	64.8	13	0.7	493	5	498	0.7	43.5	10.0
Vase-lion	57.8	22	2.0	568	28	596	3.6	56.0	11.9
Vase-lion (LEGO)	66.2	14	1.4	612	14	636	1.8	59.1	12.7

physical dimensions of LEGO voxels) and by performing only local cuts parallel to the three major planes. In this way, we can run the same algorithm in our method to optimize and generate LEGO-built internal cores, and the pyramidal residual parts for 3D printing.

Figure 20 presents physical fabrication results on the Vase-lion and Sphynx models, showing the feasibility of using LEGO bricks to build the internal core in our method. Note that, since the dimensions of standard LEGO bricks are not exactly cubical, we compose basic LEGO bricks of standard size together with flat LEGO bricks and flat LEGO tiles to form the internal cores. Table 3 presents a quantitative comparison between our method with all four building block types and our method with the LEGO bricks (cubes only). We can see from the table that for the case of LEGO-built internal cores, since it makes use of one type of bricks only, i.e., the cubes, the resulting residual volumes (V) are generally larger, so the costs of printing the residual parts ($C_{res.}$) are relatively higher. However, due to the constraints on the local cuts, we now have fewer cuts on the internal cores, so the tradeoff is that our method will generate fewer residual parts in the case of LEGO-built internal cores. Concerning the performance, it took around 25 minutes to build each of the LEGO internal cores (for an experienced LEGO builder) shown in Figure 20, while the computation time of our method for both cases (four types of building blocks vs. LEGO voxels) are similar.

Other Extensions. We may segment the input model into multiple components and construct an interior core assembly for each component; see Figure 21 (left) for an example. In addition, if we

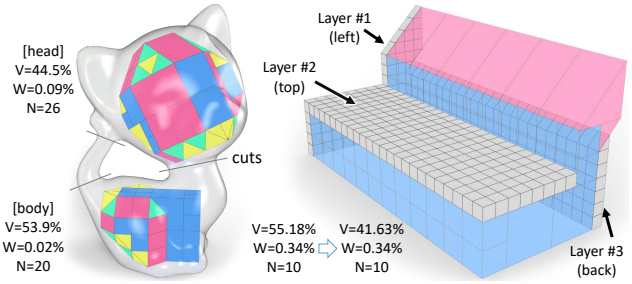


Fig. 21. Extensions: piece-wise internal core (left) and packing additional layers of half-sized building blocks (right).

utilize finer-scale building blocks, we can take a coarse decomposition result and improve the objective function by adding layers of finer-scale building blocks. Figure 21 (right) shows such an example, where we attach layers of half-sized building blocks (in gray) on the internal core, and further reduce V from 55.18% to 41.63%.

7 DISCUSSION, LIMITATION, AND FUTURE WORK

We present a novel solution, aiming at maximizing the benefits of using low-cost building blocks for cost-effective 3D fabrication. We use a universal building block set to assemble the internal core of the target 3D object, thus avoiding online fabrication of the core. By further considering pyramidal decomposition of the residual and support waste reduction, we can further minimize the necessary printing material consumption. As a result, we can reduce the overall fabrication cost and time, with the internal core constructed offline and assembled in parallel to the online printing of the residual.

The ultimate goal of this work is to enable *efficient, mass production* of 3D objects. On the one hand, the building blocks can be made in mass quantities via traditional and affordable methods such as molding. On the other hand, the universality and regularity of the block set would facilitate automated mass construction of the internal cores, e.g., using a robot along an assembly line.

Our core contribution is a computational solution that optimizes the use of a universal building block set for cost-effective 3D fabrication. We defined an objective function to seek a decomposition that leads to small residual volume, few residual parts, and low support waste. We formulated a beam search strategy to iteratively refine the internal core with local cuts, and devised methods to quickly estimate the residual waste and part count terms in the objective function, as well as pre-computation via depth peeling to further accelerate the computation. Lastly, we developed a simplified pyramidal decomposition method to partition the residual into printing-friendly (near) pyramidal parts.

As the first attempt to use building bricks for cost-effective 3D fabrication, our current approach still has some limitations. First, as we remarked above, using non-cubical and more specialized building blocks may help reduce assembly cost, but at the expense of universality. Our current optimization is geared towards cubical blocks and their derivations; an extension to more general blocks requires further investigation. Second, our solution is not scale-independent under *continuous* scaling, meaning that in general, as the input object scales up continuously, our internal core, formed

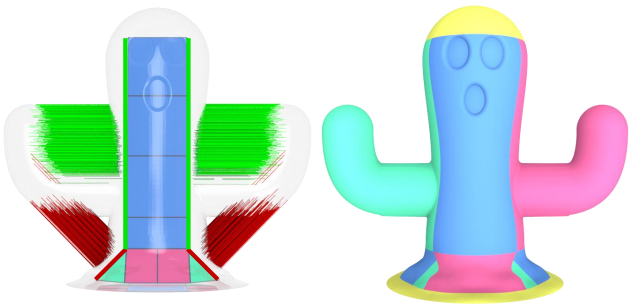


Fig. 22. An example model, where W estimated by our method (23.5%) is far larger than that by Cura (7.4%) due to the large overhangs on the left and right sides of the model. On the left, the bundled green lines show the correct estimated waste corresponding to the vertical green bases on the internal core, while the bundled red lines show the incorrect estimated waste corresponding to the red bases on the bottom. From the final decomposition shown on the right, we can see that the red wastes do not really exist.

by building blocks, cannot simply be scaled up continuously; see again Figure 15. However, if the input objects are scaled only in proportions to the minimum building block size, our method *is* scale-independent. Third, our waste estimate is an approximation. Although we have shown that the waste estimate holds merits when applied to rate the decompositions, potential miscalculations may happen due to overhangs and ignorance of the 45° and 135° corner cases. Figure 22 shows an example with large overhangs, where the waste estimated by our method (23.5%) is far larger than that by Cura (7.4%). Fourth, since we preserve the salient regions by a post-processing, the preservation is achieved only by means of selecting the right bases over a fixed internal core. Hence, we may not always avoid cutting the salient regions when partitioning the residual. Figure 8 shows the capability of our approach. We believe that the optimal approach should incorporate salient regions as hard constraints, meaning that we could, in fact, integrate it into the core solution search to avoid partitioning planes that cut through the salient regions. However, the optimization problem may then become intractable. Lastly, our current method does not explicitly consider imbalance (due to hollowed lightweight building blocks in the core), structural weaknesses, and building strength that is dictated by connections between blocks and the residuals.

In future, aside from addressing the limitations mentioned above, we would also like to study the potential of our method for fabricating large objects due to its assembly nature. On the other hand, we would like to explore the possibility of 3D printing *directly* onto an internal core assembled from building blocks, instead of attaching separately printed pieces onto it. Recent advances in the additive manufacturing community have considered the process of *building around inserts*, which corresponds to adding material to already fabricated objects, e.g., via a CNC accumulation scheme [Zhao et al. 2013]. To ensure that our internal core can be directly operated on by an additive manufacturing process, additional constraints need to be enforced, leading to an intriguing geometric problem.

ACKNOWLEDGMENTS

We thank all the anonymous reviewers for their insightful comments and feedback. We also acknowledge help from Sha He on video editing and Yuan Wei on physical fabrication. This work is supported in part by grants from National 973 Program (2015CB352501), Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK 14203416 and 14201918), Israel Science Foundation (2366/16), ISF-NSFC Joint Research Program 2217/15, NSERC grant (611370) and gift funds from Adobe.

REFERENCES

Louis Bavoil and Kevin Myers. 2008. Order Independent Transparency with Dual Depth Peeling. Tech. rep., NVIDIA Corp.

Amit H. Bermano, Thomas Funkhouser, and Szymon Rusinkiewicz. 2017. State of the Art in Methods and Representations for Fabrication-Aware Design. *Computer Graphics Forum (Eurographics)* 36, 2 (2017), 509–535. STAR volume.

Dustin Beyer, Serafima Gurevich, Stefanie Mueller, Hsiang-Ting Chen, and Patrick Baudisch. 2015. Platener: Low-fidelity Fabrication of 3D Objects by Substituting 3D Print with Laser-cut Plates *(CHI '15)*. 1799–1806.

Xuelin Chen, Hao Zhang, Jinjie Lin, Ruizhen Hu, Lin Lu, Qixing Huang, Bedrich Benes, Daniel Cohen-Or, and Baoquan Chen. 2015. Dapper: Decompose-and-pack for 3D Printing. *ACM Trans. on Graph. (SIGGRAPH Asia)* 34, 6 (2015). Article No. 213.

Michael Eigensatz, Martin Kilian, Alexander Schiftner, Niloy J. Mitra, Helmut Pottmann, and Mark Pauly. 2010. Paneling Architectural Freeform Surfaces. *ACM Trans. on Graph. (SIGGRAPH)* 29, 4 (2010). Article No. 45.

Christer Ericson. 2004. *Real-Time Collision Detection.* CRC Press.

Cass Everitt. 2001. Interactive order-independent transparency. Tech. rep., NVIDIA Corp.

Chi-Wing Fu, Chi-Fu Lai, Ying He, and Daniel Cohen-Or. 2010. K-set Tilable Surfaces. *ACM Trans. on Graph. (SIGGRAPH)* 29, 4 (2010). Article No. 44.

Philipp Herholz, Wojciech Matusik, and Marc Alexa. 2015. Approximating Free-form Geometry with Height Fields for Manufacturing. *Computer Graphics Forum (Eurographics)* 34, 2 (2015), 239–251.

John Hertz, Richard G. Palmer, and Anders S. Krogh. 1991. *Introduction to the Theory of Neural Computation* (1st ed.). Perseus Publishing.

Ruizhen Hu, Honghua Li, Hao Zhang, and Daniel Cohen-Or. 2014. Approximate Pyramidal Shape Decomposition. *ACM Trans. on Graph. (SIGGRAPH Asia)* 33, 6 (2014). Article No. 213.

Alec Jacobson. 2017. Generalized Matryoshka: Computational Design of Nesting Objects. *Computer Graphics Forum* 36, 5 (2017), 27–35.

Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. 2014. Build-to-last: Strength to weight 3D printed objects. *ACM Trans. on Graph. (SIGGRAPH)* 33, 4 (2014). Article No. 97.

Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. 2012. Chopper: partitioning models into 3D-printable parts. *ACM Trans. on Graph. (SIGGRAPH Asia)* 31, 6 (2012). Article No. 129.

Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. 2015. Legolization: Optimizing LEGO Designs. *ACM Trans. on Graph.* 34, 6, Article 222 (2015), 222:1–222:12 pages.

Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen, and Patrick Baudisch. 2014. faBrickation: Fast 3D Printing of Functional Objects by Integrating Construction Kit Building Blocks *(CHI '14)*. 3827–3834.

Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. 2013. Make it stand: balancing shapes for 3D fabrication. *ACM Trans. on Graph. (SIGGRAPH)* 32, 4 (2013). Article No. 81.

Mayank Singh and Scott Schaefer. 2010. Triangle surfaces with discrete equivalence classes. *ACM Trans. on Graph. (SIGGRAPH)* 29, 4 (2010). Article No. 46.

Peng Song, Bailin Deng, Ziqi Wang, Zhichao Dong, Wei Li, Chi-Wing Fu, and Ligang Liu. 2016. CofiFab: Coarse-to-Fine Fabrication of Large 3D Objects. *ACM Trans. on Graph. (SIGGRAPH)* 35, 4 (2016). Article No. 45.

Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. 2012. Stress relief: improving structural strength of 3D printable objects. *ACM Trans. on Graph. (SIGGRAPH)* 31, 4 (2012). Article No. 48.

Ultimaker ltd. 2017. Cura Software. https://ultimaker.com/en/products/cura-software

Juraj Vanek, JA Galicia, Bedrich Benes, R Měch, N Carr, Ondrej Stava, and GS Miller. 2014. PackMerger: A 3D Print Volume Optimizer. *Computer Graphics Forum* 33, 6 (2014), 322–332.

Lingfeng Wang and Emily Whiting. 2016. Buoyancy Optimization for Computational Fabrication. *Computer Graphics Forum (Eurographics)* 35, 2 (2016), 49–58.

Weiming Wang, Tuanfeng Y Wang, Zhouwang Yang, Ligang Liu, Xin Tong, Weihua Tong, Jiansong Deng, Falai Chen, and Xiuping Liu. 2013. Cost-effective printing of

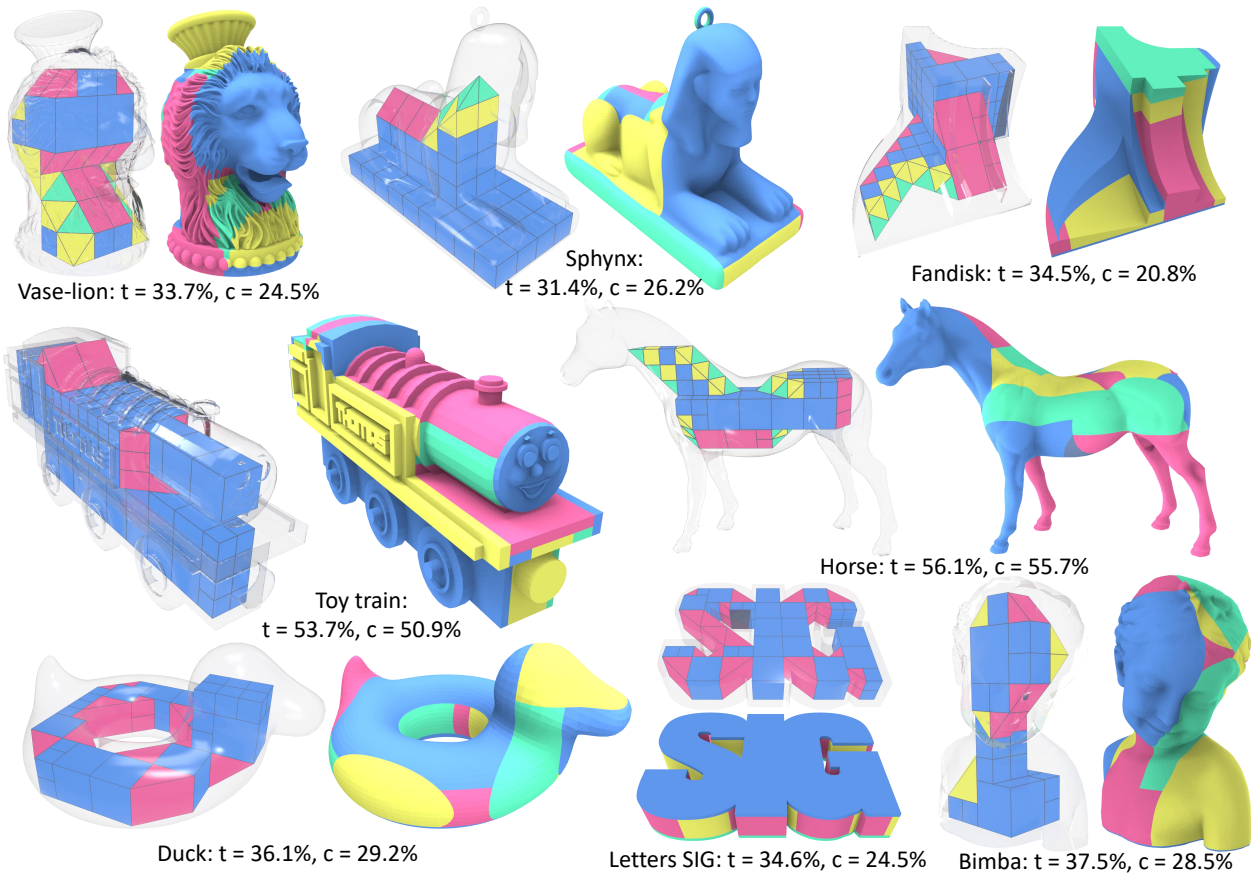


Fig. 23. A gallery of our decomposition results, showing the internal core (assembled from the universal building block set) and the pyramidal residual parts. Estimated percentage reduction in total printing time and cost are marked next to each figure for reference (obtained from Table 1).

3D objects with skin-frame structures. *ACM Trans. on Graph. (SIGGRAPH Asia)* 32, 6 (2013). Article No. 177.

Eric W. Weisstein. 2016. Space-Filling Polyhedron. From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/Space-FillingPolyhedron.html [Online; accessed 13-December-2016].

Wikipedia. 2016. Prefabrication — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Prefabrication [Online; accessed 4-December-2016].

Miaojun Yao, Zhili Chen, Linjie Luo, Rui Wang, and Huamin Wang. 2015. Level-set-based Partitioning and Packing Optimization of a Printable Model. *ACM Trans. on Graph. (SIGGRAPH Asia)* 34, 6 (2015). Article No. 214.

Xiaolong Zhang, Yang Xia, Jiaye Wang, Zhouwang Yang, Changhe Tu, and Wenping Wang. 2015. Medial axis tree - an internal supporting structure for 3D printing. *Computer Aided Geometric Design* 35 (2015), 149–162.

Xuejin Zhao, Yayue Pan, Chi Zhou, Yong Chen, and Charlie C. L. Wang. 2013. An integrated CNC accumulation system for automatic building-around-inserts. *Journal of Manufacturing Processes* 15 (2013), 432–443.

Henrik Zimmer, Florent Lafarge, Pierre Alliez, and Leif Kobbelt. 2014. Zometool shape approximation. *Graphical Models* 76, 5 (2014), 390–401.